Project #6, OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce

Vincent Rastello
rastellv@oregonstate.edu

Project #6

1. This was ran on the DGX-2 Server
    a. Each DGX server:
        i. Has 16 NVidia Tesla V100 GPUs
        ii. Has 28TB of disk, all SSD
        iii. Has two 24-core Intel Xeon 8168 Platinum 2.7GHz CPUs
        iv. Has 1.5TB of DDR4-2666 System Memory
        v. Runs the CentOS 7 Linux operating system

2.

		ARRAY MULT ADD						
	8	16	32	64	128	256	512	1024
1024	0.023	0.023	0.02	0.023	0.023	0.023	0.023	0.023
4096	0.088	0.092	0.09	0.071	0.072	0.092	0.085	0.092
8192	0.173	0.181	0.16	0.108	0.176	0.184	0.187	0.176
16384	0.344	0.352	0.249	0.362	0.366	0.369	0.288	0.361
32768	0.646	0.683	0.562	0.703	0.736	0.568	0.573	0.555
65536	1.121	1.259	1.08	1.382	1.083	1.139	1.415	1.433
131072	1.828	2.077	2.578	2.151	2.784	2.822	2.818	2.897
262144	2.639	3.687	4.424	4.942	5.382	5.271	5.46	5.44
524288	1.737	2.197	2.253	2.122	2.775	2.023	2.644	2.77
1048576	2.614	3.634	4.233	4.862	4.136	4.22	5.132	5.024
2097152	3.069	5.024	6.957	8.575	7.274	9.017	8.917	8.681
4194304	3.884	6.457	9.79	12.758	13.358	11.944	11.809	13.674
8388608	4.008	7.137	11.704	16.098	18.088	18.232	18.439	15.898
16777216	4.386	7.856	13.823	19.721	22.072	24.577	24.244	23.846
25165824	4.499	8.373	14.807	23.772	27.115	27.58	27.357	27.568

Project #6, OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce

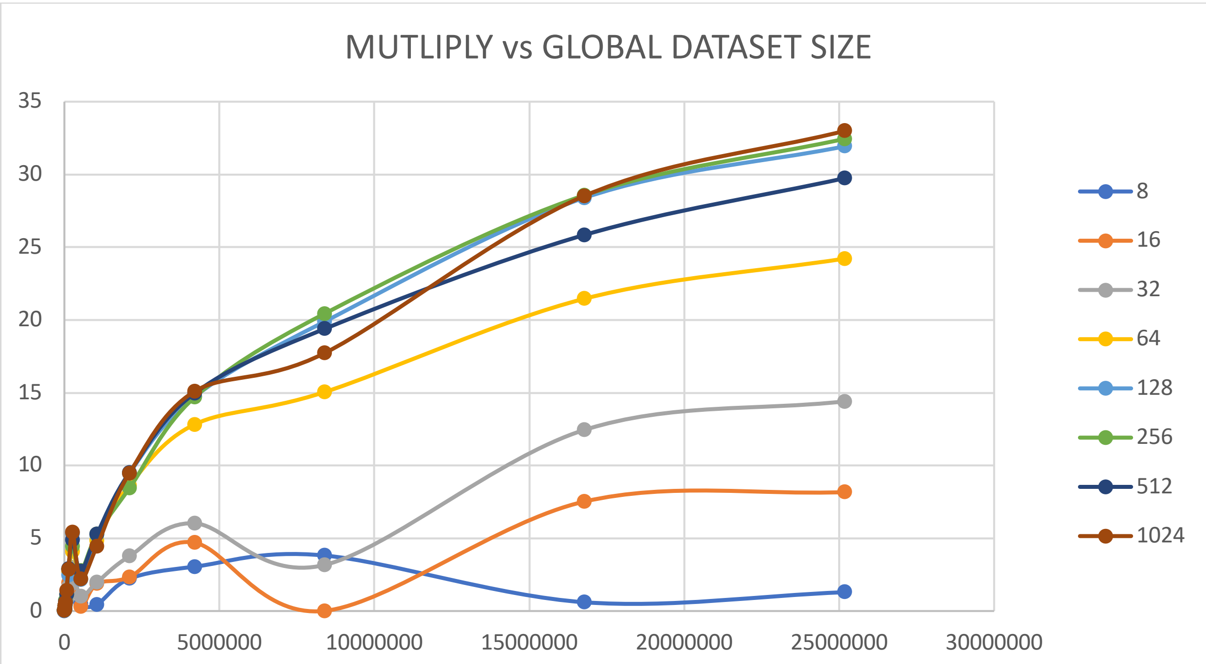
Vincent Rastello
rastellv@oregonstate.edu

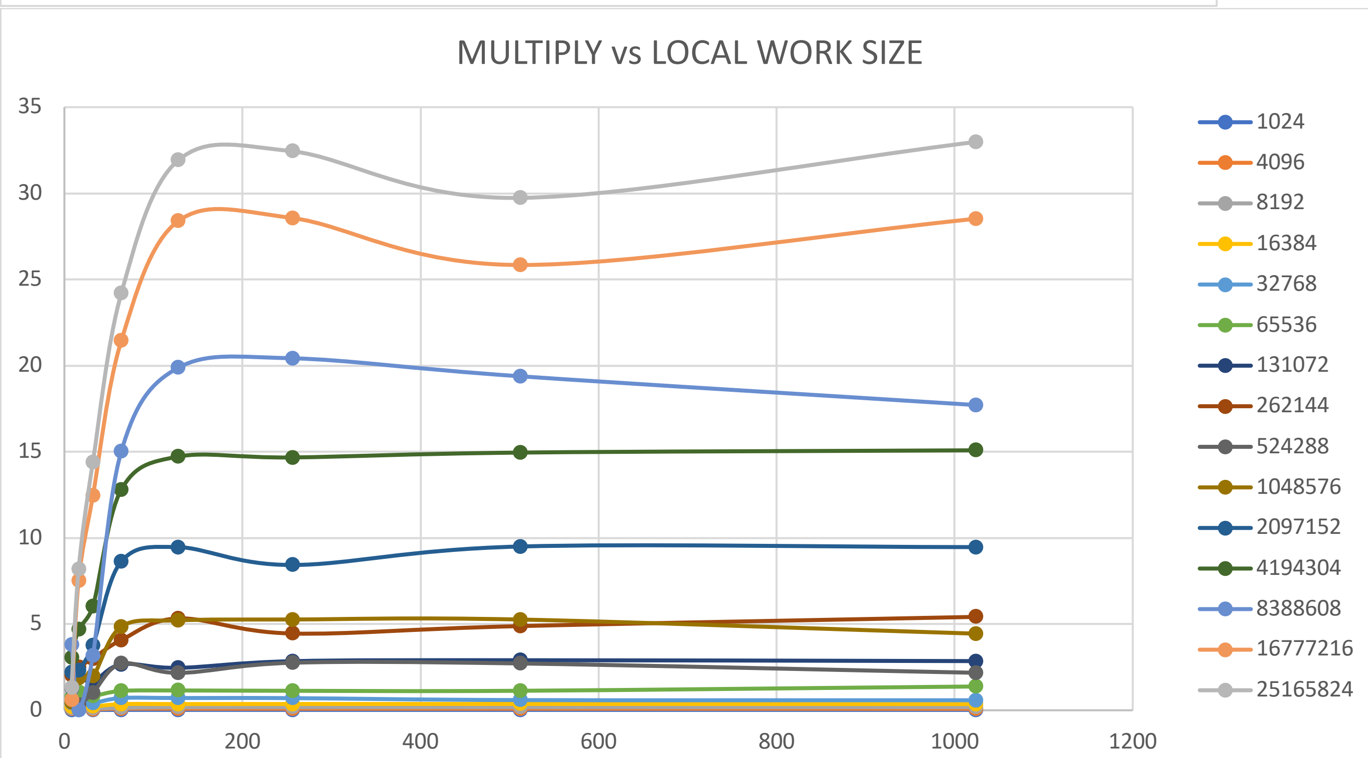
			ARRAY MULT					
	8	16	32	64	128	256	512	1024
1024	0.013	0.019	0.019	0.022	0.023	0.023	0.022	0.023
4096	0.076	0.07	0.055	0.093	0.09	0.088	0.089	0.082
8192	0.106	0.15	0.109	0.144	0.183	0.184	0.185	0.185
16384	0.213	0.263	0.218	0.358	0.36	0.364	0.367	0.36
32768	0.414	0.579	0.433	0.712	0.711	0.704	0.592	0.582
65536	0.75	1.069	0.846	1.123	1.15	1.13	1.132	1.381
131072	1.278	1.939	1.659	2.667	2.461	2.849	2.905	2.854
262144	2.021	2.519	2.995	4.078	5.331	4.455	4.886	5.42
524288	0.46	0.303	1.018	2.707	2.169	2.758	2.717	2.175
1048576	0.445	1.882	1.965	4.852	5.218	5.269	5.266	4.446
2097152	2.216	2.329	3.76	8.64	9.466	8.432	9.505	9.468
4194304	3.046	4.699	6.028	12.814	14.722	14.673	14.955	15.087
8388608	3.817	0.009	3.164	15.04	19.88	20.427	19.383	17.714
16777216	0.603	7.522	12.444	21.458	28.394	28.569	25.842	28.525
25165824	1.294	8.171	14.405	24.212	31.924	32.44	29.736	32.984

		ARRAY MULT REDUCE		
	32	64	128	256
1024	0.018	0.023	0.023	0.022
4096	0.088	0.088	0.089	0.069
8192	0.17	0.154	0.177	0.179
16384	0.35	0.358	0.343	0.353
32768	0.708	0.697	0.701	0.713
65536	1.351	1.352	1.398	1.413
131072	2.449	2.688	2.745	2.728
262144	1.267	1.367	1.048	1.431
524288	2.014	2.116	2.658	2.807
1048576	4.442	5.008	4.19	5.019
2097152	6.011	8.726	9.635	9.656
4194304	9.081	12.165	16.538	15.843
8388608	11.827	18.951	24.373	24.481
16777216	14.494	22.263	35.176	34.534
25165824	16.133	26.211	43.12	37.691

Project #6, OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce

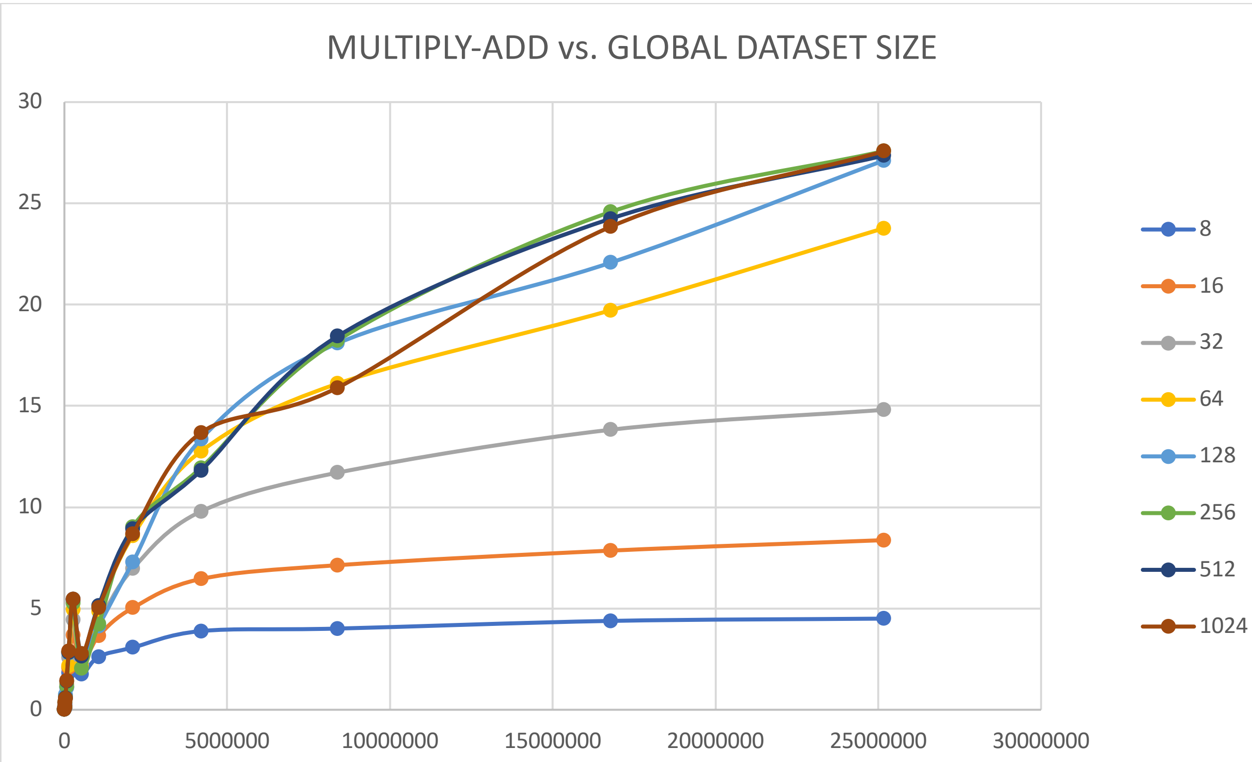
Vincent Rastello
rastellv@oregonstate.edu

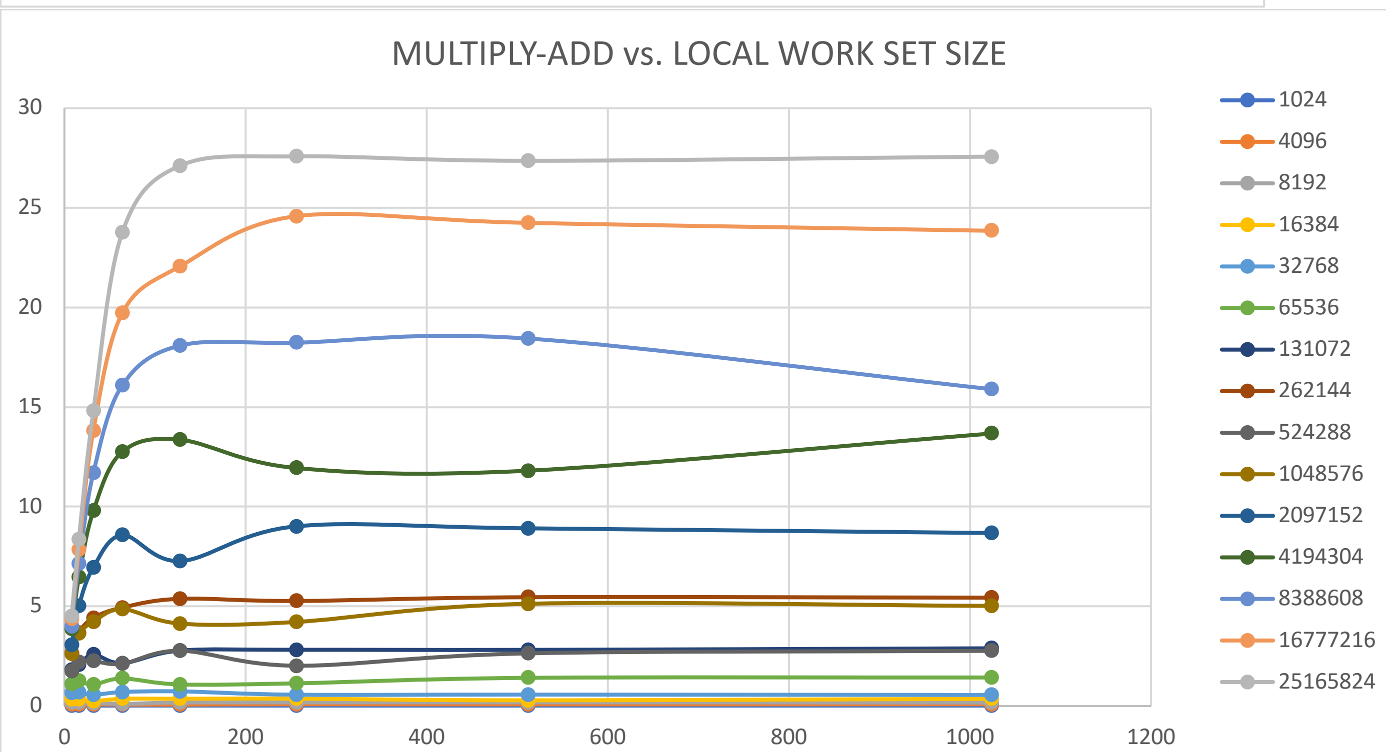




Project #6, OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce

Vincent Rastello
rastellv@oregonstate.edu

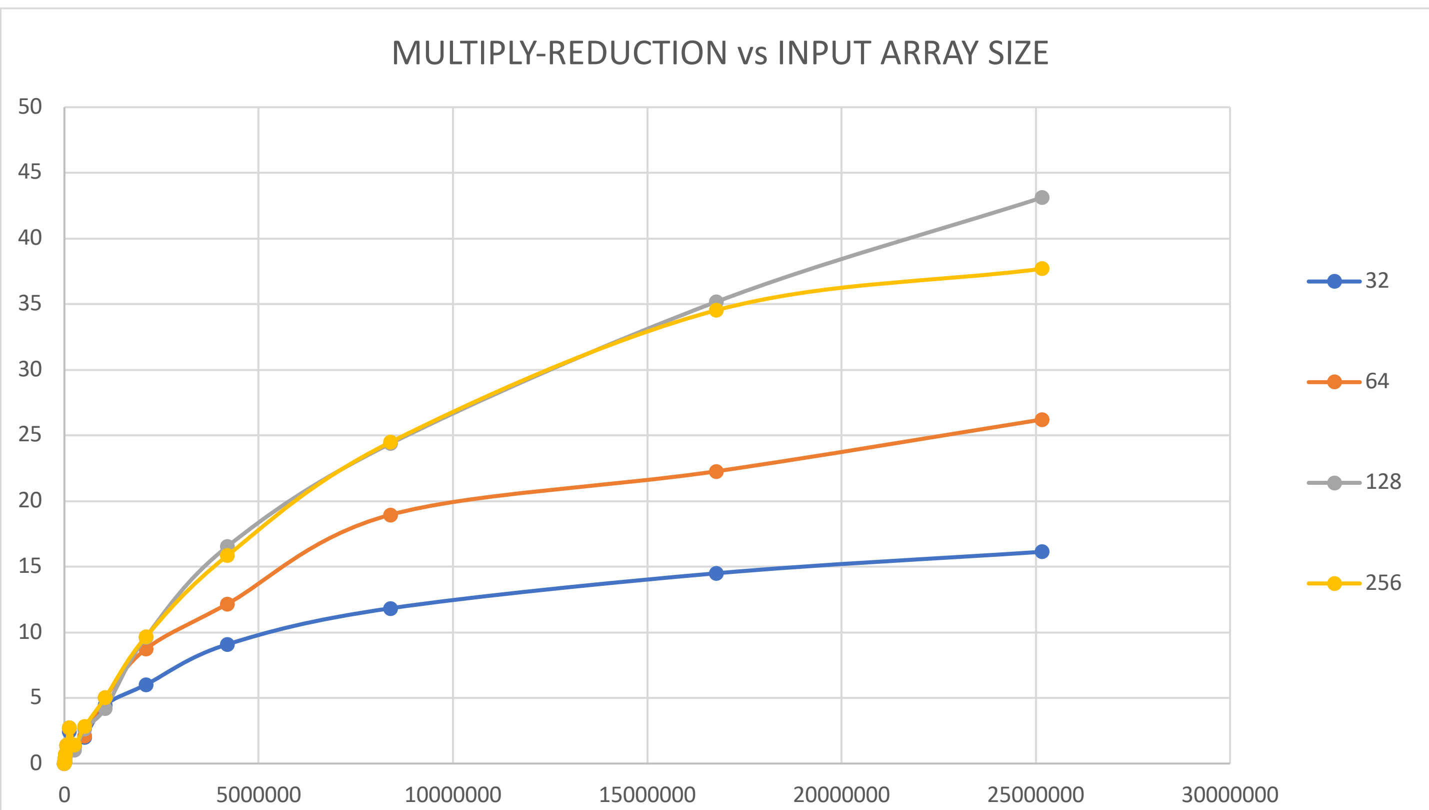




Project #6, OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce

Vincent Rastello
rastellv@oregonstate.edu

3. It looks like with the simple multiply we have more fluctuation on the lower local work size 32 and under, the simple multiply tops out at a slightly higher performance than the multiply add. The multiply add however is very close.
4. We are seeing that under work size 32 which is the minimum amount of cores per workgroup the performance takes a serious hit. This is because you are not using all possible cores resulting in cores idling. I think as the data size goes up you see a performance crash because you are waiting on the cores to become ready. As local work size increases you see a higher performance because more cores are attacking the problem. With an increase in both local work size and global data the performance increases up to 35 billion multiplies per second. This is because you are giving the machine sufficient input to satisfy it's computing power.
5. The multiply and multiply add are roughly the same because they have a fused multiply add, meaning that as the upper end of the bits are getting multiplied the lower end of the bits are being added together, so the add part is only slightly behind the multiply operation.
6. This means that with parallel GPU computing if the F parallel is high you can see huge increases in performance. The speed up is very large because the number of cores that are used is very large.



Project #6, OpenCL Array Multiply, Multiply-Add, and Multiply-Reduce

Vincent Rastello
rastellv@oregonstate.edu

2. In the multiply reduction curve we are seeing a linear relationship between local work size and performance increase, with an asymptotic relationship between performance and input data array size. Also the performance of 128 cores per work group is higher than 256.
3. The pattern for this indicates that the work size and data size increase the performance, but that performance increases due to data size eventually plateaus. The most efficient amount of cores seems to be 128, at 256 there may be too many cores to account for the amount of data resulting in inefficient use of work groups. Oddly the top performance level for the multiply reduction is higher than just the straight multiply. This may just be due to a bad run (high uptime) and then the addition of the reduction is done at log N time complexity because the additions are done by power of two. This is not that big of a change.
4. With proper use of GPU computing you can do a lot to make things more parallel friendly, the reduction in this operation adds very little negatives to performance. There is a lot that can be done with GPU computing.